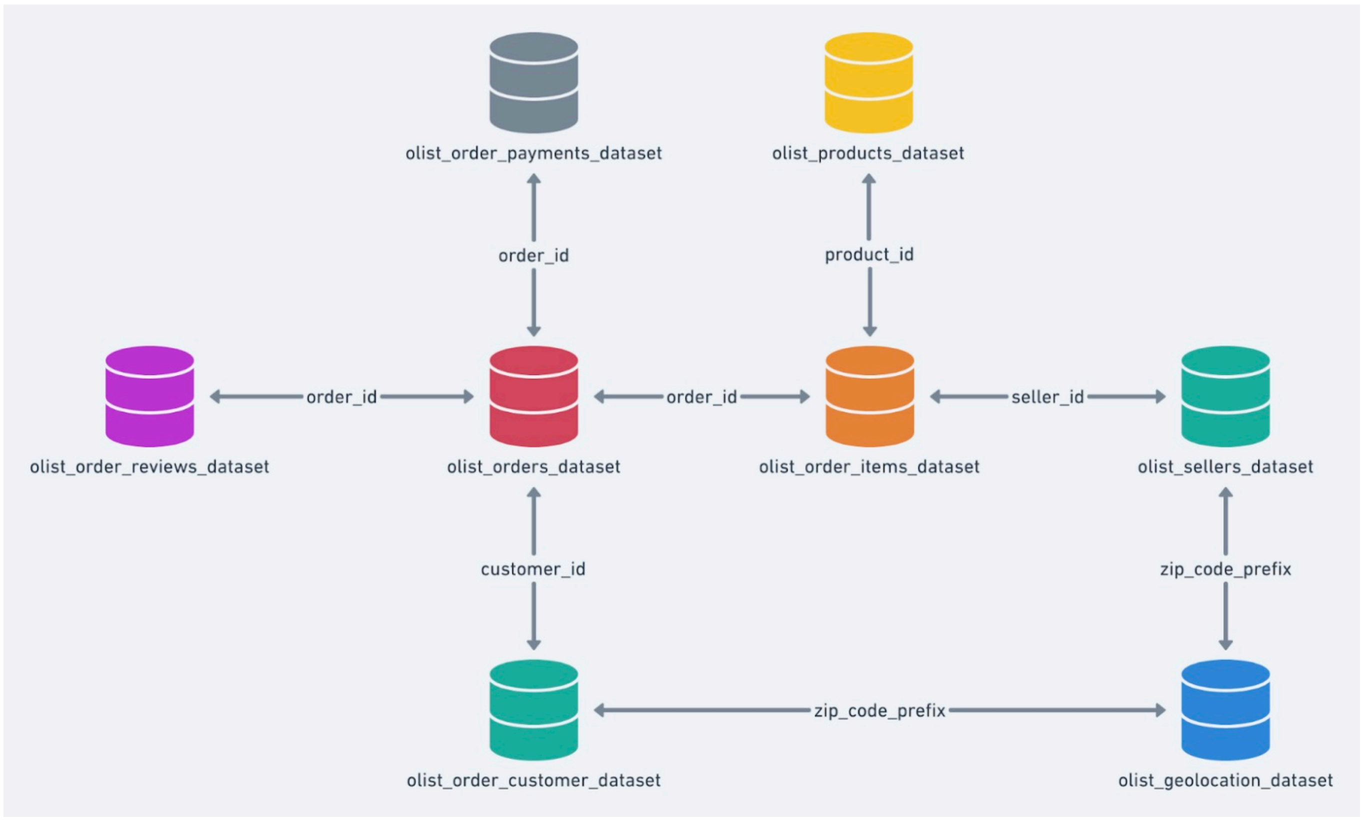
Option 2: Performance Analysis

Instructions

Dataset Overview

The Brazilian E-commerce Public Dataset by Olist provides real-world e-commerce transaction data from Brazil's largest department store, containing 100,000 orders from 2016 to 2018.

Dataset Structure



Query

```
SELECT
        o.order_id,
        o.order_status,
        o.order_purchase_timestamp,
        p.product_id,
        p.product_category_name,
        oi.price,
        oi.freight_value
FROM olist_orders_dataset o
        JOIN olist_order_items_dataset oi
```

```
                ON o.order_id = oi.order_id
        JOIN olist_products_dataset p
                ON oi.product_id = p.product_id
        JOIN olist_customers_dataset c
                ON o.customer_id = c.customer_id
WHERE
        c.customer_unique_id = '8d50f5eadf50201ccdcedfb9e2ac8455'
ORDER BY
        o.order_purchase_timestamp DESC;
```

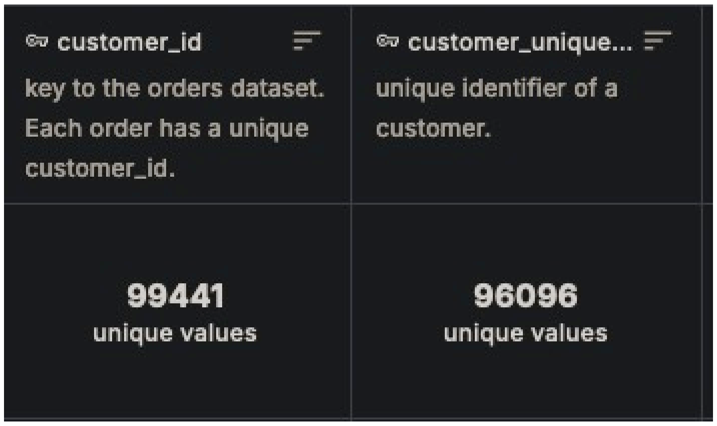
Setup

Table Creation

So the datasets we need are:

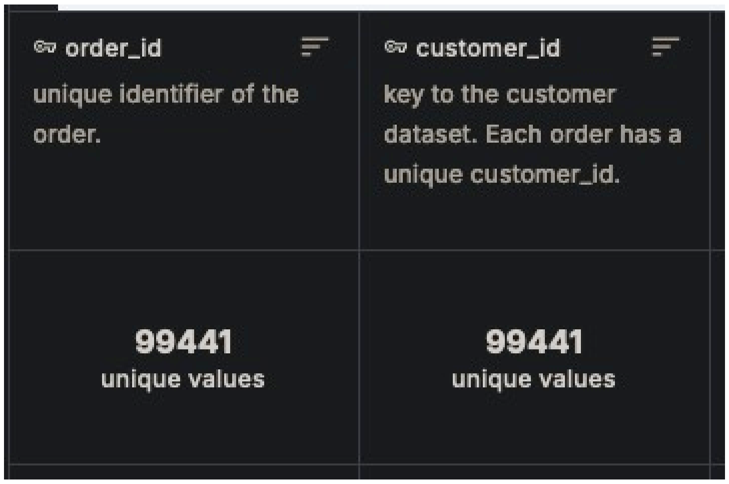
olist_orders_dataset
olist_order_items_dataset
olist_products_dataset
olist_customers_dataset

From Customers:



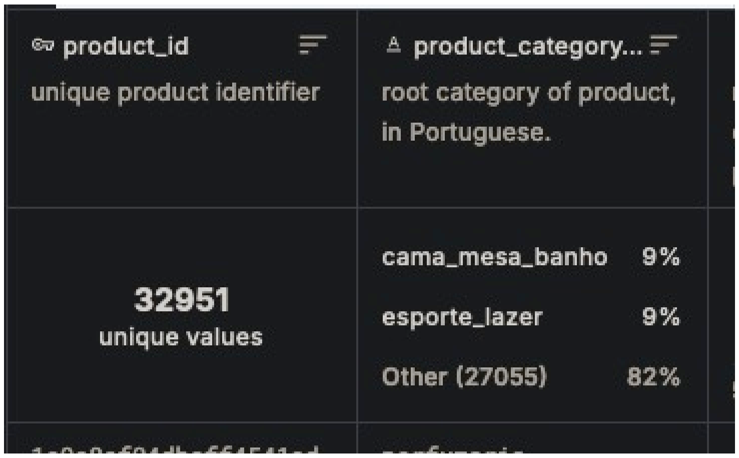
customer_id	customer_unique...
key to the orders dataset. Each order has a unique customer_id.	unique identifier of a customer.
99441 unique values	96096 unique values

From Orders:



order_id	customer_id
unique identifier of the order.	key to the customer dataset. Each order has a unique customer_id.
99441 unique values	99441 unique values

For products:

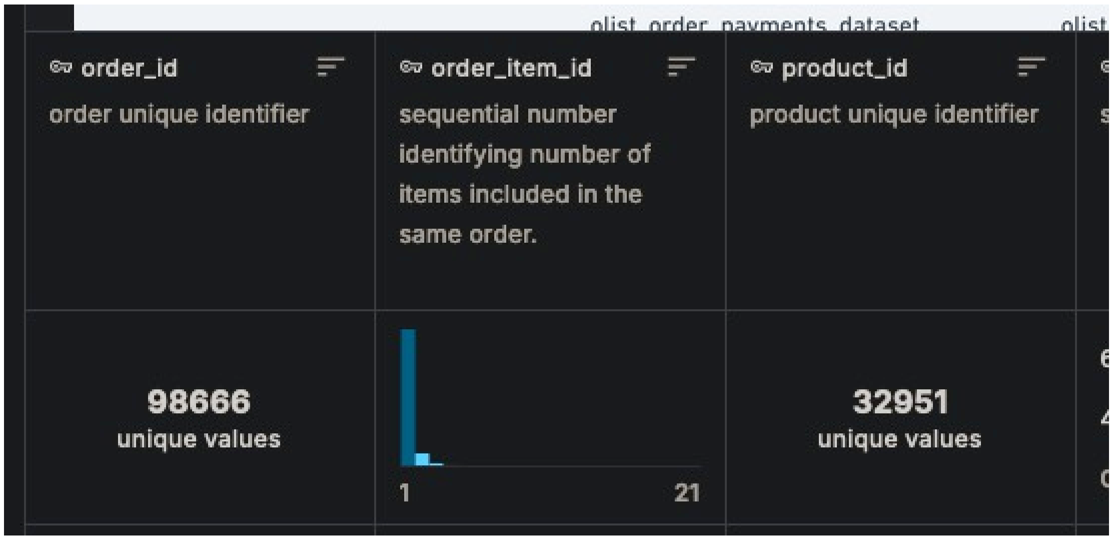


product_id
unique product identifier
32951
unique values

product_category...
root category of product, in Portuguese.

cama_mesa_banho	9%
esporte_lazer	9%
Other (27055)	82%

For order_items:



order_id
order unique identifier
98666
unique values

order_item_id
sequential number identifying number of items included in the same order.

product_id
product unique identifier
32951
unique values

So it seems that the reference goes from orders -> customers.

First we spring up the DB:

```
create schema if not exists miniproject;

use miniproject;

drop table if exists olist_order_items_dataset;
drop table if exists olist_products_dataset;
drop table if exists olist_orders_dataset;
drop table if exists olist_customers_dataset;
```

And make a table for each csv:

Do notice that some fk constraints are not here since we didn't add every table. For example: cutomer_zip_code_prefix, customer_state, and cutomer_city should probably reference the geolocation table.

```sql
create table olist_customers_dataset (
    customer_id varchar(255), -- key to orders dataset, each order has a
unique customer id.
    customer_unique_id varchar (255), -- unique identifier for a customer.
    cutomer_zip_code_prefix int, -- first five digits of customer zip code.
    cutomer_city varchar(100), -- customer city name.
    customer_state varchar(2), -- customer state.
    primary key (customer_id), -- unique id for customer_order which is what
this db represents tbh
    -- unique (customer_unique_identifier) uniqueness constraint for db
integrity; removed due to constraint failure from data
);
```

```sql
create table olist_orders_dataset (
        order_id varchar(255), -- unique identifier of the order.
    customer_id varchar(255), -- key to the customer dataset. Each order has
a unique customer_id.
    order_status varchar(100), -- reference to the order status (delivered,
shipped, etc.
    order_purchase_timestamp datetime, -- shows purchase timestamp in
format: 2017-10-02 10:56:33.
    order_approved_at datetime, -- shows the payment approval timestam in
format: 2017-10-02 10:56:33.
    order_delivered_carrier_date datetime, -- shows the order posting
timestamp. When it was handled to the logistic partner.
    order_delivered_custoemr_date datetime, -- shows the actual order
delivery date to the customer.
    order_estimated_delivery_date datetime, -- shows the estimated delivery
date that was informed to customer at the purchase moment.
    primary key (order_id), -- unique identifier
        constraint fk_customer_id foreign key (customer_id) references
olist_customers_dataset(customer_id)
);
```

```sql
create table olist_products_dataset (
        product_id varchar(255), -- unique product identifier.
    product_category_name varchar(100), -- root category of product, in
Portuguese.
    product_name_length int, -- number of characters extracted from the
product name.
    product_description_length int, -- number of characters extracted from
the product description.
    product_photos_qty int, -- number of product published photos.
    product_weight_g int, -- product weight measured in grams.
    product_length_cm int, -- product length measured in cetimeters.
    product_height_cm int, -- product height measured in cetimeters.
    product_width_cm int, -- product width measured in cetimeters.
    primary key (product_id) -- uniquely identify product
);
```

```
create table olist_order_items_dataset (
        order_id varchar(255), -- order unique identifier
    order_item_id int, -- sequential number identifying number of items
included in the same order.
    product_id varchar(255), -- product unique identifier.
    seller_id varchar(255), -- seller unique identifier. HERE WE PROLLY WANT
REFERENCE TO SELLER BUT RIP SELLER TABLE.
    shipping_limit_date datetime, -- seller shipping limit date for handling
the order over to the logistic partner in format: 2017-10-02 10:56:33.
    price decimal(10, 2), -- item price in format _.00
    freight_value decimal(10, 2), -- item freight value item in format _.00
(if an order has more than one item the freight value is splitted between
items)
    primary key (order_id, order_item_id), -- combo pk
    constraint fk_order_id foreign key (order_id) references
olist_orders_dataset(order_id),
    constraint fk_product_id foreign key (product_id) references
olist_products_dataset(product_id)
);
```

Data Insertion

Let's fix some names from the data real quick:

- product_name_lenght -> product_name_length

Some issues I fixed:
Error Code: 1062. Duplicate entry 'b6c083700ca8c135ba9f0f132930d4e8' for key 'olist_customers_dataset.customer_unique_id' -> non-unique values.

Customers

```
load data infile '/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/olist_customers_dataset.csv'
into table olist_customers_dataset
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 rows
(customer_id, customer_unique_id, cutomer_zip_code_prefix, cutomer_city,
customer_state);
```

99441 row(s) affected Records: 99441 Deleted: 0 Skipped: 0 Warnings: 0

Orders

Error Code: 1292. Incorrect date value: 'franca' for column 'order_purchase_timestamp' at row 1

Let's relax the insertion strictness and try again hoping null values or '0000' datetime objects are inserted for invalid entries.

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`miniproject`.`olist_orders_dataset`, CONSTRAINT `fk_customer_id` FOREIGN KEY (`customer_id`) REFERENCES `olist_customers_dataset` (`customer_id`))

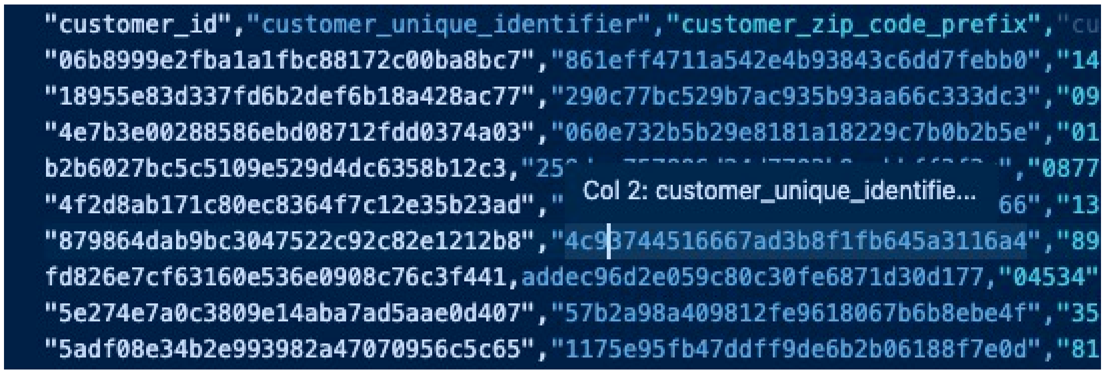
Let's check customer db:

SELECT count(customer_id) FROM miniproject.olist_customers_dataset;

'99441'

Seems like we have all the valid IDs but, upon further inspection, some ids have leading quotations and some dont.

Let's fix this for both tables, ignoring any quotation marks that happen when inserting ids.



```
"customer_id","customer_unique_identifier","customer_zip_code_prefix","cu
"06b8999e2fba1a1fbc88172c00ba8bc7","861eff4711a542e4b93843c6dd7febb0","14
"18955e83d337fd6b2def6b18a428ac77","290c77bc529b7ac935b93aa66c333dc3","09
"4e7b3e00288586ebd08712fdd0374a03","060e732b5b29e8181a18229c7b0b2b5e","01
b2b6027bc5c5109e529d4dc6358b12c3,"25                                 ","0877
"4f2d8ab171c80ec8364f7c12e35b23ad","                              66","13
"879864dab9bc3047522c92c82e1212b8","4c93744516667ad3b8f1fb645a3116a4","89
fd826e7cf63160e536e0908c76c3f441,addec96d2e059c80c30fe6871d30d177,"04534"
"5e274e7a0c3809e14aba7ad5aae0d407","57b2a98a409812fe9618067b6b8ebe4f","35
"5adf08e34b2e993982a47070956c5c65","1175e95fb47ddff9de6b2b06188f7e0d","81
```

We'll do the following for every file:

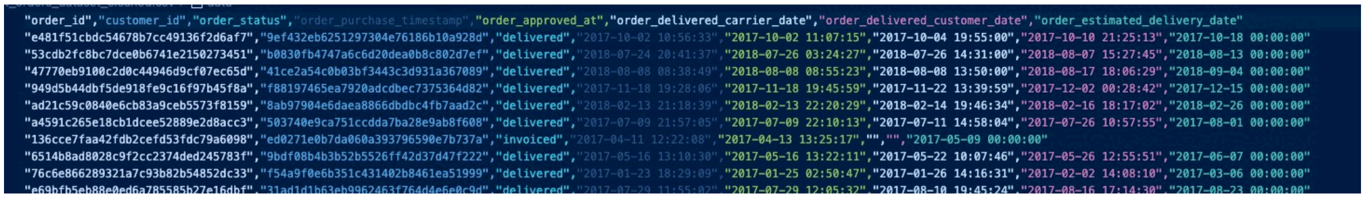
```
files = {
    "olist_customers_dataset": "/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/olist_customers_dataset.csv",
    "olist_orders_dataset": "/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/olist_orders_dataset.csv",
    "olist_products_dataset": "/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/olist_products_dataset.csv",
    "olist_order_items_dataset": "/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/olist_order_items_dataset.csv",
}

output_dir = "/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/cleaned/"

for table_name, input_path in files.items():

    df = pd.read_csv(input_path)
    output_path = f"{output_dir}{table_name}.csv"
    df.to_csv(output_path, index=False, quoting=1)
```

Cleaned versions look like:



```
"order_id","customer_id","order_status","order_purchase_timestamp","order_approved_at","order_delivered_carrier_date","order_delivered_customer_date","order_estimated_delivery_date"
"e481f51cbdc54678b7cc49136f2d6af7","9ef432eb6251297304e76186b10a928d","delivered","2017-10-02 10:56:33","2017-10-02 11:07:15","2017-10-04 19:55:00","2017-10-10 21:25:13","2017-10-18 00:00:00"
"53cdb2fc8bc7dce0b6741e2150273451","b0830fb4747a6c6d20dea0b8c802d7ef","delivered","2018-07-24 20:41:37","2018-07-26 03:24:27","2018-07-26 14:31:00","2018-08-07 15:27:45","2018-08-13 00:00:00"
"47770eb9100c2d0c44946d9cf07ec65d","41ce2a54c0b03bf3443c3d931a367089","delivered","2018-08-08 08:38:49","2018-08-08 08:55:23","2018-08-08 13:50:00","2018-08-17 18:06:29","2018-09-04 00:00:00"
"949d5b44dbf5de918fe9c16f97b45f8a","f88197465ea7920adcdbec7375364d82","delivered","2017-11-18 19:28:06","2017-11-18 19:45:59","2017-11-22 13:39:59","2017-12-02 00:28:42","2017-12-15 00:00:00"
"ad21c59c0840e6cb83a9ceb5573f8159","8ab97904e6daea8866dbdbc4fb7aad2c","delivered","2018-02-13 21:18:39","2018-02-13 22:20:29","2018-02-14 19:46:34","2018-02-16 18:17:02","2018-02-26 00:00:00"
"a4591c265e18cb1dcee52889e2d8acc3","503740e9ca751ccdda7ba28e9ab8f608","delivered","2017-07-09 21:57:05","2017-07-09 22:10:13","2017-07-11 14:58:04","2017-07-26 10:57:55","2017-08-01 00:00:00"
"136cce7faa42fdb2cefd53fdc79a6098","ed0271e0b7da060a393796590e7b737a","invoiced","2017-04-11 12:22:08","2017-04-13 13:25:17","","","2017-05-09 00:00:00"
"6514b8ad8028c9f2cc2374ded245783f","9bdf08b4b3b52b5526ff42d37d47f222","delivered","2017-05-16 13:10:30","2017-05-16 13:22:11","2017-05-22 10:07:46","2017-05-26 12:55:51","2017-06-07 00:00:00"
"76c6e866289321a7c93b82b54852dc33","f54a9f0e6b351c431402b8461ea51999","delivered","2017-01-23 18:29:09","2017-01-25 02:50:47","2017-01-26 14:16:31","2017-02-02 14:08:10","2017-03-06 00:00:00"
"e69bfb5eb88e0ed6a785585b27e16dbf","31ad1d1b63eb9962463f764d4e6e0c9d","delivered","2017-07-29 11:55:02","2017-07-29 12:05:32","2017-08-10 19:45:24","2017-08-16 17:14:30","2017-08-23 00:00:00"
```

Even after doing this we get the same error. So we are going to create a temporary table and only insert relevant values:

```
drop table if exists temp_olist_orders_dataset;

create temporary table temp_olist_orders_dataset like olist_orders_dataset;

load data infile '/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/cleaned/olist_orders_dataset.csv'
into table temp_olist_orders_dataset
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 rows
(order_id, customer_id, order_status, order_purchase_timestamp,
order_approved_at, order_delivered_carrier_date,
order_delivered_custoemr_date, order_estimated_delivery_date);

insert into olist_orders_dataset
select *
from temp_olist_orders_dataset
where customer_id in (select customer_id from olist_customers_dataset);

select * from olist_orders_dataset;

select count(order_id) from olist_orders_dataset;
```

Somehow, we get all relevant orders: 'count(order_id)' : '99441'

So we can move on

Products

Error Code: 1366. Incorrect integer value: '' for column 'product_name_length' at row 106

Why did I see this coming?

Well that row looks like:

```
"a41e356c76fab66334f36de622ecbd3a","","","","","650.0","17.0","14.0","12.0"
```

Since we are already preprocessing with pandas. Let's fix these values too to become default 0.

```
file_path = '/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/olist_products_dataset.csv'
output_path = '/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/cleaned/olist_products_dataset.csv'

df = pd.read_csv(file_path)

numeric_columns = [
    'product_name_length',
    'product_description_length',
    'product_photos_qty',
    'product_weight_g',
    'product_length_cm',
    'product_height_cm',
    'product_width_cm'
]

for col in numeric_columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

df.fillna({'product_name_length': 0,
           'product_description_length': 0,
           'product_photos_qty': 0,
           'product_weight_g': 0,
           'product_length_cm': 0,
           'product_height_cm': 0,
           'product_width_cm': 0},
           inplace=True)

df.to_csv(output_path, index=False, quoting=1)
```

```
"a41e356c76fab66334f36de622ecbd3a","","0.0","0.0","0.0","650.0","17.0","14.0
","12.0"
```

Nice:

```
load data infile '/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/cleaned/olist_products_dataset.csv'
into table olist_products_dataset
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 rows
(product_id, product_category_name, product_name_length,
product_description_length, product_photos_qty, product_weight_g,
product_length_cm, product_height_cm, product_width_cm);
```

32951 row(s) affected Records: 32951 Deleted: 0 Skipped: 0 Warnings: 0

Order Items

I wont even write the query since I know it's already going to fail cs of the numeric columns.

Let's do another preprocess:

```
file_path = '/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/olist_order_items_dataset.csv'
output_path = '/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/cleaned/olist_order_items_dataset.csv'

df = pd.read_csv(file_path)

# Probably should've done this before for other columns, will look at this
later
df['shipping_limit_date'] = pd.to_datetime(df['shipping_limit_date'],
errors='coerce')

numeric_columns = ['order_item_id', 'price', 'freight_value']
for col in numeric_columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

df.fillna({'order_item_id': 0,
           'price': 0.00,
           'freight_value': 0.00},
           inplace=True)

df.to_csv(output_path, index=False, quoting=1)
```

And now try to insert:

```
load data infile '/Users/suape/WorkDir/Main Vault/Classes +
Uni/PDS/MiniProject/DataSource/cleaned/olist_order_items_dataset.csv'
into table olist_order_items_dataset
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 rows
(order_id, order_item_id, product_id, seller_id, shipping_limit_date, price,
freight_value);
```

112650 row(s) affected Records: 112650 Deleted: 0 Skipped: 0 Warnings: 0

Nice

Analysis

Query

```sql
SELECT
        o.order_id,
        o.order_status,
        o.order_purchase_timestamp,
        p.product_id,
        p.product_category_name,
        oi.price,
        oi.freight_value
FROM olist_orders_dataset o
        JOIN olist_order_items_dataset oi
                ON o.order_id = oi.order_id
        JOIN olist_products_dataset p
                ON oi.product_id = p.product_id
        JOIN olist_customers_dataset c
                ON o.customer_id = c.customer_id
WHERE
        c.customer_unique_id = '8d50f5eadf50201ccdcedfb9e2ac8455'
ORDER BY
        o.order_purchase_timestamp DESC;
```

Output

- Run EXPLAIN ANALYZE on the baseline query
- Record:
  - Execution time
  - Number of rows processed
- Save the execution plan

I actually ran into this problem in a previous assignment where me and my team had to process millions of rows, so I will also run explain on this:

Explain:

id	select_type	table	partitions	type	possible_keys	key
1	SIMPLE	c	NULL	ALL	PRIMARY	NULL

id	select_type	table	partitions	type	possible_keys	key
1	SIMPLE	o	NULL	ref	PRIMARY,fk_customer_id	fk_customer_ic
1	SIMPLE	oi	NULL	ref	PRIMARY,fk_product_id	PRIMARY
1	SIMPLE	p	NULL	eq_ref	PRIMARY	PRIMARY

Explain Analyze:

```
-> Sort: o.order_purchase_timestamp DESC  (actual time=46..46 rows=16
loops=1)\n    -> Stream results  (cost=18732 rows=9000) (actual
time=5.06..45.9 rows=16 loops=1)\n        -> Nested loop inner join
(cost=18732 rows=9000) (actual time=5.05..45.8 rows=16 loops=1)\n
-> Nested loop inner join  (cost=15582 rows=9000) (actual time=5.03..45.7
rows=16 loops=1)\n              -> Nested loop inner join  (cost=12432
rows=9000) (actual time=5.01..45.4 rows=17 loops=1)\n                  ->
Filter: (c.customer_unique_id = \'8d50f5eadf50201ccdcedfb9e2ac8455\')
(cost=9282 rows=9000) (actual time=4.95..45 rows=17 loops=1)\n
-> Table scan on c  (cost=9282 rows=90004) (actual time=0.218..36.7
rows=99441 loops=1)\n                  -> Index lookup on o using
fk_customer_id (customer_id=c.customer_id)  (cost=0.25 rows=1) (actual
time=0.0248..0.0253 rows=1 loops=17)\n              -> Filter:
(oi.product_id is not null)  (cost=0.25 rows=1) (actual time=0.0121..0.0132
rows=0.941 loops=17)\n                  -> Index lookup on oi using
PRIMARY (order_id=o.order_id)  (cost=0.25 rows=1) (actual
time=0.0116..0.0126 rows=0.941 loops=17)\n          -> Single-row index
lookup on p using PRIMARY (product_id=oi.product_id)  (cost=0.25 rows=1)
(actual time=0.00958..0.00961 rows=1 loops=16)\n
```

Let's consult: https://planetscale.com/blog/how-read-mysql-explains

> In MySQL 8.0.18, `EXPLAIN ANALYZE` was introduced, a new concept built on top of the regular `EXPLAIN` query plan inspection tool. In addition to the query plan and estimated costs, which a normal `EXPLAIN` will print, `EXPLAIN ANALYZE` also prints the *actual* costs of individual iterators in the execution plan.
> `EXPLAIN ANALYZE` actually runs the query, so if you don't want to run the query against your live database, do not use `EXPLAIN ANALYZE`.

Right, so, let's do a quick replacement of the \n with actual new lines using python:

```
Execution Plan:
-> Sort: o.order_purchase_timestamp DESC  (actual time=46..46 rows=16
loops=1)
    -> Stream results  (cost=18732 rows=9000) (actual time=5.06..45.9
rows=16 loops=1)
        -> Nested loop inner join  (cost=18732 rows=9000) (actual
time=5.05..45.8 rows=16 loops=1)
            -> Nested loop inner join  (cost=15582 rows=9000) (actual
time=5.03..45.7 rows=16 loops=1)
                -> Nested loop inner join  (cost=12432 rows=9000) (actual
time=5.01..45.4 rows=17 loops=1)
                    -> Filter: (c.customer_unique_id =
'8d50f5eadf50201ccdcedfb9e2ac8455')  (cost=9282 rows=9000) (actual
time=4.95..45 rows=17 loops=1)
                        -> Table scan on c  (cost=9282 rows=90004) (actual
time=0.218..36.7 rows=99441 loops=1)
                    -> Index lookup on o using fk_customer_id
(customer_id=c.customer_id)  (cost=0.25 rows=1) (actual time=0.0248..0.0253
rows=1 loops=17)
                -> Filter: (oi.product_id IS NOT NULL)  (cost=0.25 rows=1)
(actual time=0.0121..0.0132 rows=0.941 loops=17)
                    -> Index lookup on oi using PRIMARY
(order_id=o.order_id)  (cost=0.25 rows=1) (actual time=0.0116..0.0126
rows=0.941 loops=17)
            -> Single-row index lookup on p using PRIMARY
(product_id=oi.product_id)  (cost=0.25 rows=1) (actual time=0.00958..0.00961
rows=1 loops=16)
```

Analysis

From both queries we can see that:

- The total execution time is 46 ms
- We process:
    - olist_customers_dataset: 99,441 rows
        - olist_orders_dataset: 17 rows
        - olist_order_items_dataset: 17 rows
        - olist_products_dataset: 16 rows

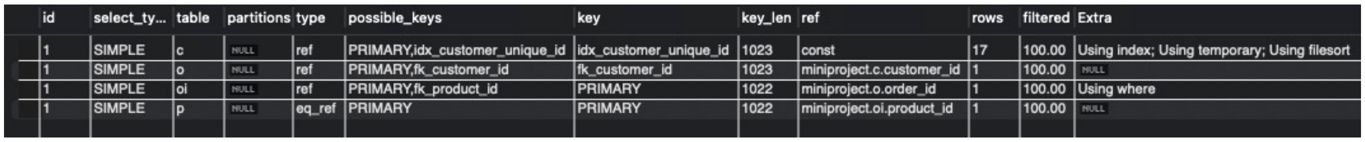
Since we use primary keys of orders, orderitems, customers, and products, there were automatically generated indexes for us.

The absence of an index on customer_unique_id forces MySQL to scan all 99,441 rows to evaluate the filter condition c.customer_unique_id = '8d50f5eadf50201ccdcedfb9e2ac8455'.

Since we did not make this the pk given the data specs, we have to create an index for this too. This will prevent the full table scan from taking place.

```
create index idx_customer_unique_id on
olist_customers_dataset(customer_unique_id);
```

Much better now:



id	select_ty...	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	ref	PRIMARY,idx_customer_unique_id	idx_customer_unique_id	1023	const	17	100.00	Using index; Using temporary; Using filesort
1	SIMPLE	o	NULL	ref	PRIMARY,fk_customer_id	fk_customer_id	1023	miniproject.c.customer_id	1	100.00	NULL
1	SIMPLE	oi	NULL	ref	PRIMARY,fk_product_id	PRIMARY	1022	miniproject.o.order_id	1	100.00	Using where
1	SIMPLE	p	NULL	eq_ref	PRIMARY	PRIMARY	1022	miniproject.oi.product_id	1	100.00	NULL

```
-> Sort: o.order_purchase_timestamp DESC  (actual time=2.21..2.22 rows=16
loops=1)
    -> Stream results  (cost=23.8 rows=17) (actual time=0.342..2.15 rows=16
loops=1)
        -> Nested loop inner join  (cost=23.8 rows=17) (actual
time=0.332..2.1 rows=16 loops=1)
            -> Nested loop inner join  (cost=17.8 rows=17) (actual
time=0.278..1.66 rows=16 loops=1)
                -> Nested loop inner join  (cost=11.9 rows=17) (actual
time=0.219..1.06 rows=17 loops=1)
                    -> Covering index lookup on c using
idx_customer_unique_id
(customer_unique_id=''8d50f5eadf50201ccdcedfb9e2ac8455'')  (cost=5.9
rows=17) (actual time=0.101..0.117 rows=17 loops=1)
                    -> Index lookup on o using fk_customer_id
(customer_id=c.customer_id)  (cost=0.256 rows=1) (actual time=0.053..0.0547
rows=1 loops=17)
                -> Filter: (oi.product_id is not null)  (cost=0.256 rows=1)
(actual time=0.0323..0.0345 rows=0.941 loops=17)
                    -> Index lookup on oi using PRIMARY
(order_id=o.order_id)  (cost=0.256 rows=1) (actual time=0.0318..0.0339
rows=0.941 loops=17)
            -> Single-row index lookup on p using PRIMARY
(product_id=oi.product_id)  (cost=0.256 rows=1) (actual time=0.0269..0.027
rows=1 loops=16)
```

Now, the query uses a covering index lookup on customer_unique_id, resulting in a significant improvement.

Instead of scanning the whole table, it directly retrieves the matching rows using the index.

Can we do better?

Well, this was the main bottleneck, but notice that we are doing an order by at the end of the query. This can be optimized by creating a composite index on olist_orders_dataset(order_purchase_timestamp, order_id). Sorting will use the index directly instead of creating temporary tables, saving some computation time.

```
create index idx_order_timestamp on
olist_orders_dataset(order_purchase_timestamp, order_id);
```

```
-> Sort: o.order_purchase_timestamp DESC  (actual time=1.45..1.45 rows=16
loops=1)
    -> Stream results  (cost=24.9 rows=19.5) (actual time=0.212..1.41
rows=16 loops=1)
        -> Nested loop inner join  (cost=24.9 rows=19.5) (actual
time=0.202..1.37 rows=16 loops=1)
            -> Nested loop inner join  (cost=18 rows=19.5) (actual
time=0.178..1.09 rows=16 loops=1)
                -> Nested loop inner join  (cost=11.9 rows=17) (actual
time=0.145..0.721 rows=17 loops=1)
                    -> Covering index lookup on c using
idx_customer_unique_id
(customer_unique_id=''8d50f5eadf50201ccdcedfb9e2ac8455'')  (cost=5.9
rows=17) (actual time=0.0796..0.0916 rows=17 loops=1)
                    -> Index lookup on o using fk_customer_id
(customer_id=c.customer_id)  (cost=0.256 rows=1) (actual time=0.035..0.0365
rows=1 loops=17)
                -> Filter: (oi.product_id is not null)  (cost=0.257
rows=1.15) (actual time=0.0194..0.0212 rows=0.941 loops=17)
                    -> Index lookup on oi using PRIMARY
(order_id=o.order_id)  (cost=0.257 rows=1.15) (actual time=0.019..0.0207
rows=0.941 loops=17)
            -> Single-row index lookup on p using PRIMARY
(product_id=oi.product_id)  (cost=0.255 rows=1) (actual time=0.0167..0.0168
rows=1 loops=16)
```

We can see that the actual time now goes down by around 30%. At scale, even though it was a few seconds, it is helpful!